```
/*
TinyTrackGPS.cpp - A simple track GPS to SD card logger.
TinyTrackGPS v0.10

Copyright © 2019-2021 Francisco Rafael Reyes Carmona.
All rights reserved.

rafael.reyes.carmona@gmail.com

  This file is part of TinyTrackGPS.

  TinyTrackGPS is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  TinyTrackGPS is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with TinyTrackGPS.  If not, see <https://www.gnu.org/licenses/>.
*/

/*****************************************************************************
/  Programa de localizacion por gps que graba las posiciones en
/  un fichero de texto cada segundo, de forma diaria.
/
/  - Conectar módulo SD con pin CS (naranja) en pin 10 arduino.
/
/  Uso de librería TinyGPS.
/   Requiere uso de librería SoftwareSerial, se presupone que disponemos
/   de un dispositivo GPS serie de 9600-bauds conectado en pines 9(rx) y 8(tx).
/  - Conectar módulo NMEA-6M (gps) pines 8,9 (9 - pin rx negro)
/
/  - Conectar LCD 16x2 pines 2,3,4,5,6,7 (2-amarillo , 3-azul,
/     4-rojo, 5-azul oscuro, 6-verde, 7-blanco)
/
/  - Conectar OLED 0.96" en SDA y SCL. pines A4 y A5 del Arduino UNO.
*******************************************************************************/

// Include libraries.
#include <Arduino.h>
#include "config.h"
#include "Display.h"
//#include <SoftwareSerial.h>
#include "TinyGPS_fixed.h"
#if defined(__LGT8F__) && defined(nop)
#undef nop
#endif
#include <SdFat.h>
#include <sdios.h>
#include <UTMConversion.h>
#include <Timezone.h>

// Definimos el Display
#if defined(DISPLAY_TYPE_LCD_16X2)
Display LCD(LCD_16X2);
```

```
#elif defined(DISPLAY_TYPE_LCD_16X2_I2C)
Display LCD(LCD_16X2_I2C);
#elif defined(DISPLAY_TYPE_SDD1306_128X64)
Display LCD(SDD1306_128X64);
#elif defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
Display LCD(SDD1306_128X64);
#else
#define NO_DISPLAY
#include <LowPower.h>
#endif

// Variables para grabar en SD.
char GPSLogFile[] = "YYYYMMDD.csv"; // Formato de nombre de fichero. YYYY-Año, MM-Mes, DD-Día.

const uint8_t CHIP_SELECT = SS;  // SD card chip select pin. (10)
SdFat card;   //SdFat.h library.
SdFile file;
bool SDReady;
bool SaveOK;

// Variables y clases para obtener datos del GPS y conversion UTM.
TinyGPS gps;
GPS_UTM utm;
//SoftwareSerial gps_serial(9, 8);
#define gps_serial Serial
int year_gps;
byte month_gps, day_gps, hour_gps, minute_gps, second_gps;
float flat, flon;
unsigned long age;
unsigned int elev;

// Central European Time (Frankfurt, Paris) See below for other zone.
TimeChangeRule CEST = {"CEST", Last, Sun, Mar, 2, 120};     // Central European Summer Time
TimeChangeRule CET = {"CET ", Last, Sun, Oct, 3, 60};       // Central European Standard Time
Timezone CE(CEST, CET);
#define TimeZone CE

// Variables para gestionar el tiempo local.
TimeElements time_gps;
time_t utctime;
time_t localtime;
time_t prevtime;

/*
-------------------------------------------------------------------------------------
 * Info for timezone:

// Australia Eastern Time Zone (Sydney, Melbourne)
TimeChangeRule aEDT = {"AEDT", First, Sun, Oct, 2, 660};    // UTC + 11 hours
TimeChangeRule aEST = {"AEST", First, Sun, Apr, 3, 600};    // UTC + 10 hours
Timezone ausET(aEDT, aEST);
#define TimeZone ausET

// Moscow Standard Time (MSK, does not observe DST)
TimeChangeRule msk = {"MSK", Last, Sun, Mar, 1, 180};
Timezone tzMSK(msk);
```

```
#define TimeZone tzMSK

// Central European Time (Frankfurt, Paris)
TimeChangeRule CEST = {"CEST", Last, Sun, Mar, 2, 120};     // Central European Summer Time
TimeChangeRule CET = {"CET ", Last, Sun, Oct, 3, 60};       // Central European Standard Time
Timezone CE(CEST, CET);
#define TimeZone CE

// United Kingdom (London, Belfast)
TimeChangeRule BST = {"BST", Last, Sun, Mar, 1, 60};        // British Summer Time
TimeChangeRule GMT = {"GMT", Last, Sun, Oct, 2, 0};         // Standard Time
Timezone UK(BST, GMT);
#define TimeZone UK

// UTC
TimeChangeRule utcRule = {"UTC", Last, Sun, Mar, 1, 0};     // UTC
Timezone UTC(utcRule);
#define TimeZone UTC

// US Eastern Time Zone (New York, Detroit)
TimeChangeRule usEDT = {"EDT", Second, Sun, Mar, 2, -240};  // Eastern Daylight Time = UTC - 4 hours
TimeChangeRule usEST = {"EST", First, Sun, Nov, 2, -300};   // Eastern Standard Time = UTC - 5 hours
Timezone usET(usEDT, usEST);
#define TimeZone usET

// US Central Time Zone (Chicago, Houston)
TimeChangeRule usCDT = {"CDT", Second, Sun, Mar, 2, -300};
TimeChangeRule usCST = {"CST", First, Sun, Nov, 2, -360};
Timezone usCT(usCDT, usCST);
#define TimeZone usCT

// US Mountain Time Zone (Denver, Salt Lake City)
TimeChangeRule usMDT = {"MDT", Second, Sun, Mar, 2, -360};
TimeChangeRule usMST = {"MST", First, Sun, Nov, 2, -420};
Timezone usMT(usMDT, usMST);
#define TimeZone usMT

// Arizona is US Mountain Time Zone but does not use DST
Timezone usAZ(usMST);
#define TimeZone usAZ

// US Pacific Time Zone (Las Vegas, Los Angeles)
TimeChangeRule usPDT = {"PDT", Second, Sun, Mar, 2, -420};
TimeChangeRule usPST = {"PST", First, Sun, Nov, 2, -480};
Timezone usPT(usPDT, usPST);
#define TimeZone usPT
----------------------------------------------------------------------------------------
*/

//------------------------------------------------------------------------------
/*
 * User provided date time callback function.
 * See SdFile::dateTimeCallback() for usage.
 */
void dateTime(uint16_t* date, uint16_t* time) {
  // User gets date and time from GPS or real-time
  // clock in real callback function

  // return date using FAT_DATE macro to format fields
  //*date = FAT_DATE(year, month, day);
  *date = (year(localtime)-1980) << 9 | month(localtime) << 5 | day(localtime);

  // return time using FAT_TIME macro to format fields
  //*time = FAT_TIME(hour, minute, second);
  *time = hour(localtime) << 11 | minute(localtime) << 5 | second(localtime) >> 1;
}
//--------------------------------------------------------------------------------

void GPSData(TinyGPS &gps, GPS_UTM &utm);
#ifndef NO_DISPLAY
void ScreenPrint(Display &LCD, TinyGPS &gps, GPS_UTM &utm);
#ifndef DISPLAY_TYPE_SDD1306_128X64
bool pinswitch();
#endif
#endif
//void GPSRefresh();
#if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
unsigned long iteration = 0;
#endif

void setup(void) {
  #if defined(__LGT8F__)
  ECCR = 0x80;
  ECCR = 0x00;
  #endif
  //Serial.begin(9600);
  gps_serial.begin(9600);

  //Serial.print(F("Initializing SD card..."));

  SDReady = card.begin(CHIP_SELECT);
  //(SDReady) ? Serial.println(F("Done.")) : Serial.println(F("FAILED!"));

  /* Iniciaización del display LCD u OLED */
  #ifndef NO_DISPLAY
  LCD.start();
  //LCD.clr();

  #endif

  //Serial.print(F("Waiting for GPS signal..."));
  #ifndef NO_DISPLAY
  //LCD.clr();
  #if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C) ||
defined(DISPLAY_TYPE_SDD1306_128X64)
  LCD.print(NAME, VERSION, "Waiting for ","GPS signal...");
  #elif defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
  #if defined(__LGT8F__)
  //LCD.print(NAME_M, VERSION);
  LCD.DrawLogo();
  #else
  LCD.print(NAME_M, VERSION);
  #endif
  #endif
  unsigned int time = 0;
  #endif

  bool config = false;

  do {
    #ifndef NO_DISPLAY
    LCD.wait_anin(time++);
    #endif
    for (unsigned long start = millis(); millis() - start < 1000;) {
      while (gps_serial.available() > 0) {
        char c = gps_serial.read();
        //Serial.write(c); // uncomment this line if you want to see the GPS data flowing
        if (gps.encode(c)) {// Did a new valid sentence come in?
          gps.crack_datetime(&year_gps, &month_gps, &day_gps, &hour_gps, &minute_gps, &second_gps, NULL, &age);
          (age != TinyGPS::GPS_INVALID_AGE) ? config = true : config = false;
        }
      }
    }
  }while(!config);

  time_gps.Year = year_gps - 1970;
  time_gps.Month = month_gps;
  time_gps.Day = day_gps;
  time_gps.Hour = hour_gps;
  time_gps.Minute = minute_gps;
  time_gps.Second = second_gps;
  utctime = makeTime(time_gps);
  localtime = TimeZone.toLocal(utctime);
  prevtime = utctime;
  //Serial.println(F("Done."));
  //Serial.println(F("Configuration ended."));
  #ifndef NO_DISPLAY
  LCD.clr();
  #endif
}

void loop(void) {
  bool gps_ok = false;

  while (gps_serial.available() > 0) {
    char c = gps_serial.read();
    //Serial.write(c); // uncomment this line if you want to see the GPS data flowing
    if (gps.encode(c)) {
      gps.crack_datetime(&year_gps, &month_gps, &day_gps, &hour_gps, &minute_gps, &second_gps, NULL, &age);
      gps_ok = true;
    }
  }

  gps.f_get_position(&flat, &flon, &age);
  if ((elev = gps.altitude()) == TinyGPS::GPS_INVALID_ALTITUDE) elev = 0;
  else elev /= 100L;
  utm.UTM(flat, flon);

  time_gps.Year = year_gps - 1970;
  time_gps.Month = month_gps;
  time_gps.Day = day_gps;
  time_gps.Hour = hour_gps;
  time_gps.Minute = minute_gps;
  time_gps.Second = second_gps;
  utctime = makeTime(time_gps);
  localtime = TimeZone.toLocal(utctime);

  if (gps_ok) {
    if (utctime > prevtime) {
      GPSData(gps, utm);
      prevtime = utctime;
      #if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
      iteration++;
      #endif
    }
    #ifndef NO_DISPLAY
    ScreenPrint(LCD, gps, utm);
    #endif
  }
  // Este código no hace verdaderamente ahorrar energía. Consume más que si no lo uso.
  //LowPower.idle(SLEEP_12MS, ADC_OFF, TIMER2_OFF, TIMER1_OFF, TIMER0_OFF, SPI_ON, USART0_ON, TWI_ON);
  //
  #ifdef NO_DISPLAY
  LowPower.powerSave(SLEEP_250MS, ADC_OFF, BOD_ON,TIMER2_OFF); // para NO_DISPLAY.
  #endif
}

void GPSData(TinyGPS &gps, GPS_UTM &utm) {
  static char buffer[62];
  static char line[11];
  static int index;
  static bool save;

  if (age != TinyGPS::GPS_INVALID_AGE){
    index = snprintf(buffer,10, "%02d:%02d:%02d,", hour(localtime), minute(localtime), second(localtime));
    dtostrf(flat, 10, 6, line);
    index += snprintf(buffer+index,12,"%s,",line);
    dtostrf(flon, 10, 6, line);
    index += snprintf(buffer+index,12,"%s,",line);
    index += snprintf(buffer+index,7,"%05u,",elev);
    index += snprintf(buffer+index,19,"%02d%c %ld %ld", utm.zone(), utm.band(), utm.X(), utm.Y());
    //Serial.print(buffer);
  }

  sprintf(GPSLogFile, "%04d%02d%02d.csv", year(localtime), month(localtime), day(localtime));

  //SdFile::dateTimeCallback(dateTime);
  FsDateTime::setCallback(dateTime);



  // Si no existe el fichero lo crea y añade las cabeceras.
  if (SDReady && !card.exists(GPSLogFile)) {
    if (file.open(GPSLogFile, O_CREAT | O_APPEND | O_WRITE)) {
      //Serial.print(F("New GPSLogFile, adding heads..."));
      file.println(F("Time, Latitude, Longitude, Elevation, UTM Coords (WGS84)"));
      //Serial.println(F("Done."));
      file.close();
      }
      //else {
      //Serial.println(F("** Error creating GPSLogFile. **"));
      //}
  }
  if (SDReady && (save = file.open(GPSLogFile, O_APPEND | O_WRITE))) {
    //Serial.print(F("Open GPSLogFile to write..."));
    file.println(buffer);
    file.close();
    //Serial.println(F("Done."));
  } else {
    //Serial.println(F("** Error opening GPSLogFile. **"));
  }
  //} //else Serial.println(F("** GPS signal lost. **"));
  SaveOK = save;
}

#ifndef NO_DISPLAY
void ScreenPrint(Display &LCD, TinyGPS &gps, GPS_UTM &utm){
  bool print_utm = false;
  bool print_grades = false;
  static unsigned short sats;

  sats = gps.satellites();
  #if defined(DISPLAY_TYPE_SDD1306_128X64) || defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
  //if (LCD.display_type() == SDD1306_128X64) {
    print_utm = true;
    print_grades = true;
  //}
  #endif
  #if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
  if (!pinswitch()) print_utm = true;
  else print_grades = true;
  #endif

  if (print_utm) {
    static char line[12];
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    sprintf(line, "%02d%c?%ld?", utm.zone(), utm.band(), utm.X());
    #else
    sprintf(line, "%02d%c %ld ", utm.zone(), utm.band(), utm.X());
    #endif
    //Serial.println(line);
    LCD.print(0,0,line);
    LCD.print_PChar((byte)6);
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    sprintf(line, "%02hu?", sats);
    #else
    sprintf(line, "%02hu ", sats);
    #endif
    //Serial.println(line);
    LCD.print(12,0,line);
    if (SaveOK) LCD.print_PChar((byte)7);
    else LCD.print("-");

    // New line
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    sprintf(line, "%ld?", utm.Y());
    #else
    sprintf(line, "%ld ", utm.Y());
    #endif
    //Serial.println(line);
    LCD.print(1,1,line);
    LCD.print_PChar((byte)5);
    #if defined(DISPLAY_TYPE_SDD1306_128X64_lcdgfx)
    sprintf(line, "%u@", elev);
    #else
    sprintf(line, "%um", elev);
    #endif
    //Serial.println(line);

    if (elev < 10) LCD.print(14,1,line);
    else if (elev < 100) LCD.print(13,1,line);
    else if (elev < 1000) LCD.print(12,1,line);
    else LCD.print(11,1,line);
  }

  if (print_grades) {
    /*
    #ifndef DISPLAY_TYPE_SDD1306_128X64
    LCD.print(0,0," ");
    LCD.print(15,0," ");
    //LCD.print(0,1," ");
    LCD.print(15,1," ");
    #endif
    */
    static char line[11];
    LCD.print(1,(LCD.display_type() == SDD1306_128X64) ? 2 : 0,"LAT/");
    dtostrf(flat, 8, 6, line);
    LCD.print(line);
    LCD.print(1,(LCD.display_type() == SDD1306_128X64) ? 3 : 1,"LON/");
    dtostrf(flon, 8, 6, line);
    LCD.print(line);
  }
}

#if defined(DISPLAY_TYPE_LCD_16X2) || defined(DISPLAY_TYPE_LCD_16X2_I2C)
bool pinswitch() {
  bool pin;

  pin = bitRead(iteration,4); // Change every 8 seconds.
  //LCD.clr(); -> Too slow clear individual characters.
  if ((iteration%16) == 0) {
    LCD.print(0,0," ");
    LCD.print(15,0," ");
    //LCD.print(0,1," ");
    LCD.print(15,1," ");
  }
  return pin;
}
#endif
#endif
/*
void GPSRefresh()
{
    while (gps_serial.available() > 0)
      gps.encode(gps_serial.read());
```

```
}
*/
/*
time_t makeTime_elements(int year, byte month, byte day, byte hour, byte minute, byte second){
  static TimeElements tm;

  tm.Year = year - 1970;
  tm.Month = month;
  tm.Day = day;
  tm.Hour = hour;
  tm.Minute = minute;
  tm.Second = second;

  return makeTime(tm);
}
*/
```